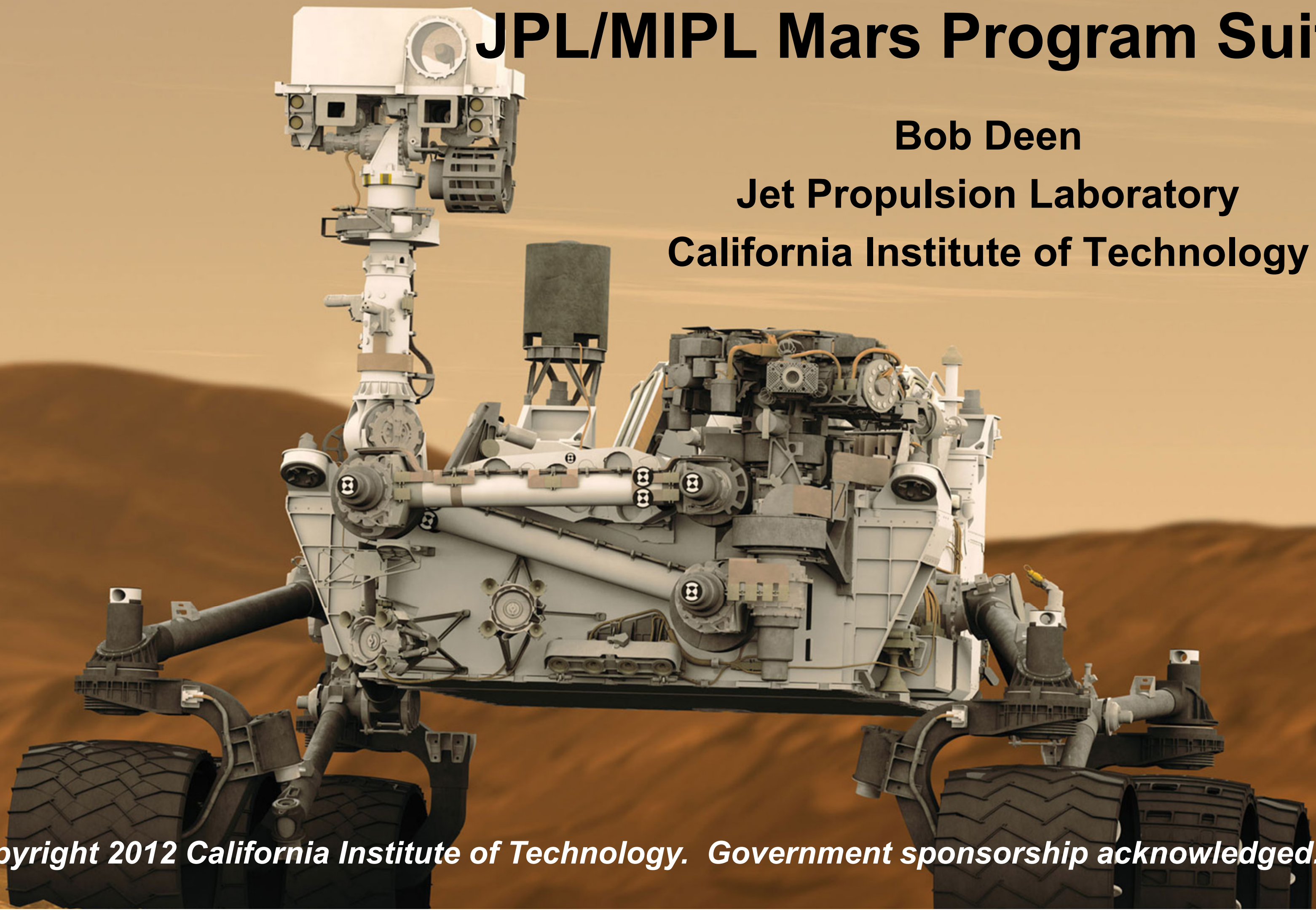
JPL Multimission Instrument Processing Laboratory (MIPL)

Software Reuse in the Planetary Context: the JPL/MIPL Mars Program Suite

Bob Deen

Jet Propulsion Laboratory

California Institute of Technology



Background

- **Multimission Instrument (Image) Processing Lab at MIPL**
- **Responsible for the ground-based instrument data processing for (among other things) all recent in-situ Mars missions:**
  - Mars Pathfinder
  - Mars Polar Lander (MPL)
  - Mars Exploration Rovers (MER)
  - Phoenix
  - Mars Science Lab (MSL)
- **Responsibilities for in-situ missions**
  - Reconstruction of instrument data from telemetry
  - Systematic creation of Reduced Data Records (RDRs) for images
  - Creation of special products for operations, science, and public outreach
  - In the critical path for operations
    - MIPL products required for planning the next Sol's activities

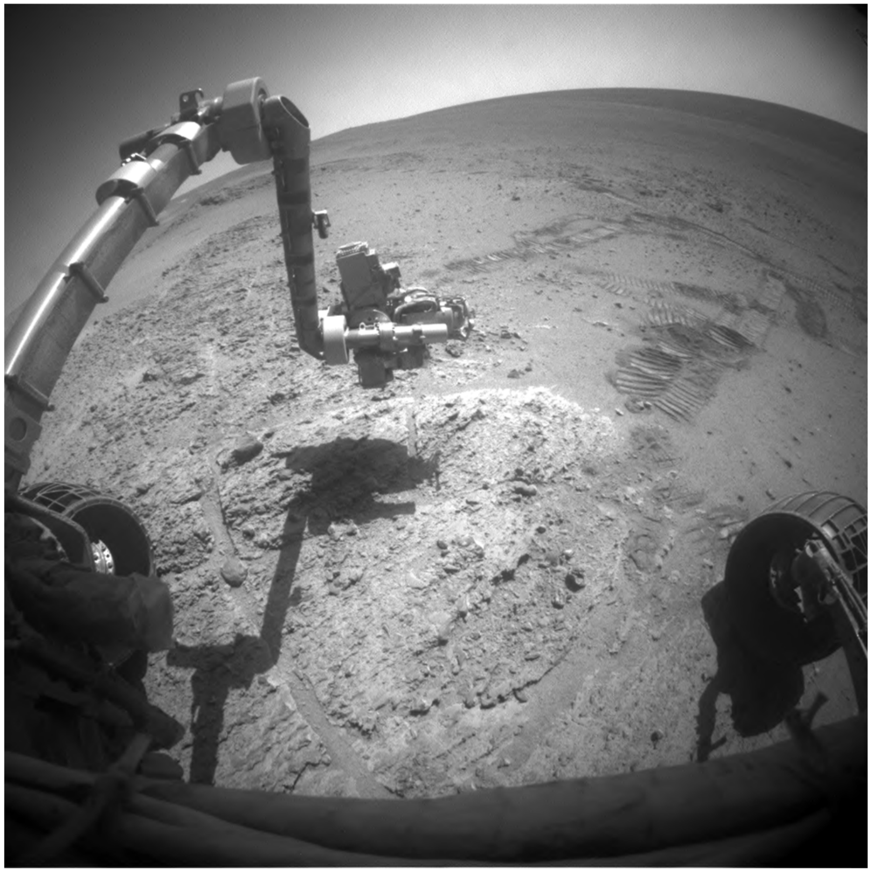
Product Categories

- **Tactical Products**
  - Used for daily operations
  - Critical path for rover operations
    - Rover Planners (drivers)
    - Science Planners (defining targets and goals)
  - Tight timing requirements
    - 1-30 minutes, depending on product
- **Strategic Products**
  - Long-term rover operational planning (days to weeks)
  - Science users
  - Public release

Product Types

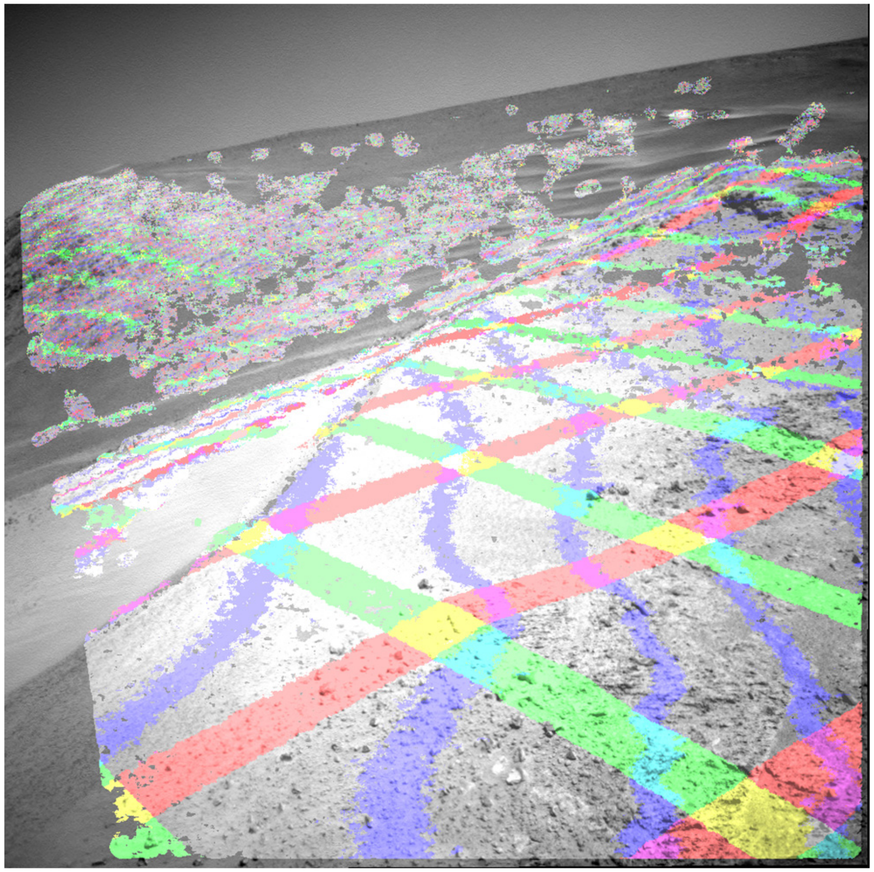
- **Over two dozen distinct products per stereo pair**
  - Double that if you include L->R and R->L
- **Key products:**
  - Radiometrically corrected images
  - Geometrically rectified images
  - Disparity maps
  - XYZ images
  - Surface normals
  - Slope maps
  - Reachability/Preload maps (for arm instruments)
  - Roughness maps
- **Multiple-image products**
  - Terrain Meshes
    - Including orbital meshes
  - Mosaics

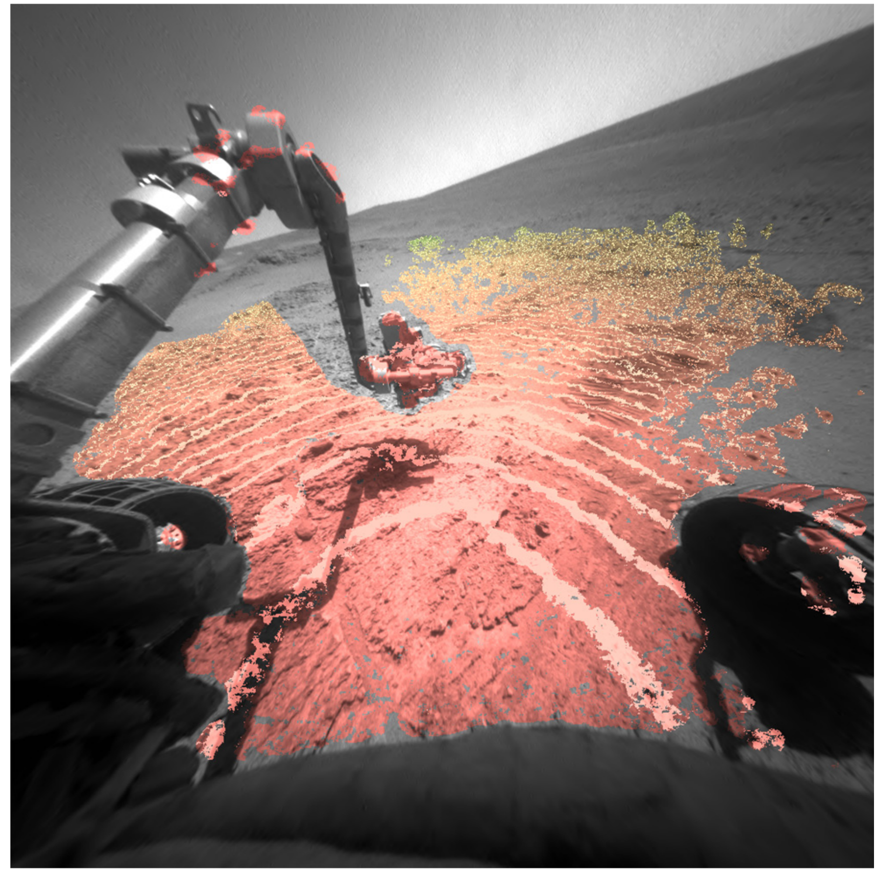
Raw and Linearized Image



Opportunity front hazcam, sol 2819. Raw on left, linearized on right

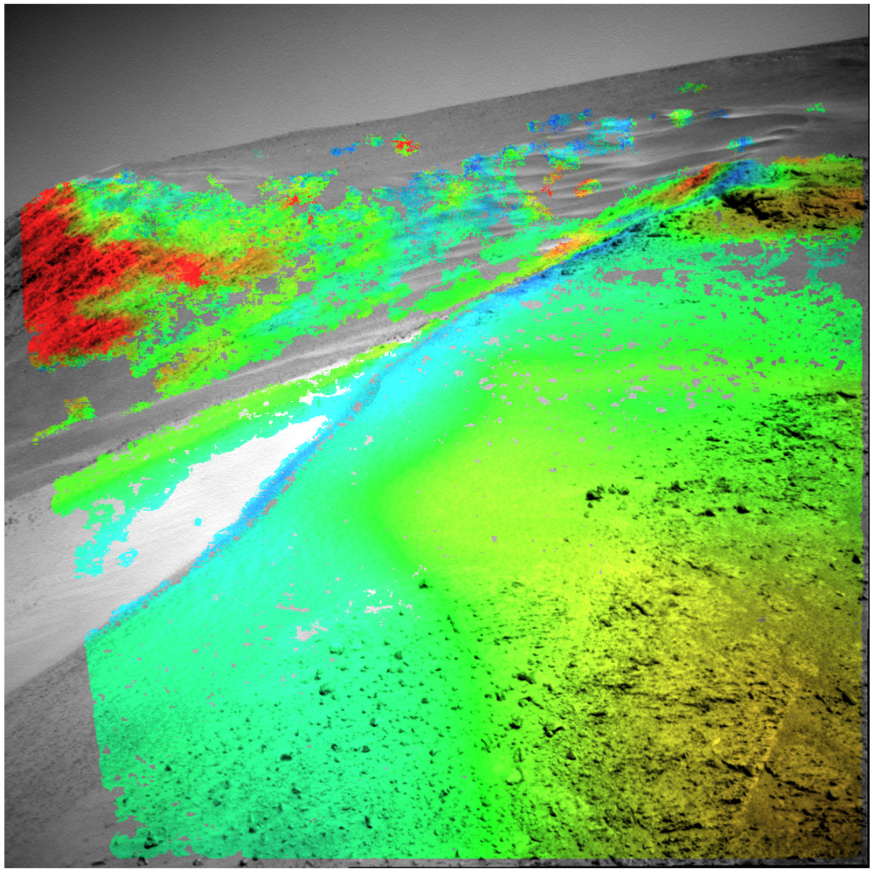
XYZ and Range Image

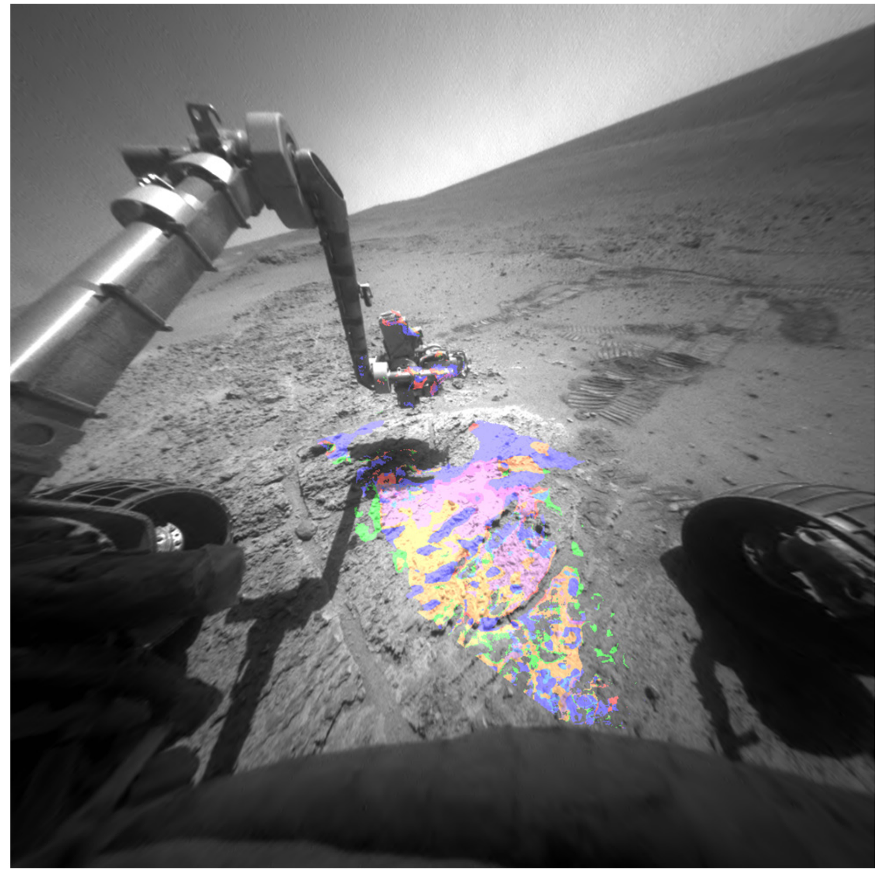




Left: Opportunity navcam, sol 2820; XYZ shows lines of constant X (red) and Y (green) at 1m spacing, with constant Z (blue) at 0.1m.
Right: Front hazcam, sol 2819; range has 1m spacing

Slope and Reachability Image





Left: Slope from navcam, sol 2820. Colors indicate slope; 0-20 degrees is blue->red. Right: Arm reachability from front hazcam, sol 2819. Colors indicate different instruments or arm configurations.

Software History

- **Scope: RDR-generation programs (except wedge/mesh)**
  - **Collectively called the Mars program suite**
- **Development started for Mars Pathfinder in 1994**
  - MPF-specific programs
  - Hard-coded parameters, inflexible algorithms, code repetition
- **Software suite rewritten for Mars Polar Lander**
  - Analysis indicated significant commonality between missions
    - Future missions deemed likely to want similar capabilities
  - Reusable, mission-independent design
- **Significant upgrades in capability for each new mission (MER, PHX, MSL)**
  - Yet mission adaptation remains relatively simple

Software Design Overview

- **Set of 43 application programs**
  - All but 4 are multimission
    - No mission-specific code
  - Multimission exceptions
    - Arm reachability for MER, PHX, MSL
      - Uses flight software, insufficient commonality across missions
    - MSL rover mask
      - Uses flight software to create mask based on kinematic state
      - First mission doing this; may abstract in future
- **Built on VICAR image processing system**
  - Core infrastructure: image I/O, parameter processing, O/S isolation
  - Very mature
- **Mission-specific aspects encapsulated into a library**
  - Planetary Image Geometry (PIG)

PIG Library

- **Object-oriented C++ class library**
- **Abstracts most functionality needed for in-situ missions (rover and lander) into base classses**
  - Camera model, pointing, coordinate systems, metadata access, etc.
- **Subclasses contain mission dependencies**
  - How to point the MER navcam
  - What a MSL image label looks like
  - How to remove dark current on a PHX image
- **Seven missions currently supported**
  - MPF, Mars 01 (testbed), FIDO (testbed), Generic, MER, Phoenix, MSL
  - MPL has been obsoleted
  - Software also used for Moonrise and InSight proposal demos and LSOT testbed
- **New missions added easily**
  - Each amounts to only 5-6% of the code base
  - Adaptation time ~1/20 of time needed to write original library

Adaptation Experiences

- **Adaptation times vary from 2 days to a few months**
  - Hard to estimate in many cases
    - Have to separate adaptation from adding new functionality
  - Compare to 3 years to write original code
- **MPL/MPF**
  - Adaptation done together, along with core library development
  - About 6 weeks for MPL, 3 weeks for MPF
- **Testbeds**
  - Mars ‘01 Athena testbed: 2 days (actual measurement)
  - Moonrise/InSight/LSOT: About a week
  - FIDO development rover: ~3 weeks
- **Generic “mission”: About a week**
- **MER: About two months**
- **Phoenix: About two months**
- **MSL: About three months**

Lines of Code

Component	Approx. Lines of Code
PIG Library (Total)	47100
PIG Multimission Base	22800
PIG MPL	2400
PIG M01	1200
PIG Generic	1200
PIG FIDO	3000
PIG MER	4900
PIG PHX	6100
PIG MSL	5500
Applications	99800

Primary Classes

- **PigModelBase: Common services, parameter and error handling**
- **PigMission: Factory methods to create all other objects based on metadata**
- **PigCameraModel: Translate line/sample to/from XYZ vector in 3- space**
  - Subclasses for different model types: CAHV, CAHVOR, CAHVORE, etc
- **PigPointingModel: Mission- and instrument-specific articulation of camera**
- **PigSurfaceModel: Describe the ground, and intersect view rays with it**
  - Subclasses for different model types: plane, sphere, infinity, etc
- **PigFileModel: High-level access to metadata and image data**
- **PigLabelModel: Facilities for mission-specific output file metadata**
- **PigCoordSystem: Conversion between coordinate systems**
- **PigSite: Define position of moveable object, such as rover, at a given instant**
- **RadiometryModel: Describe how to correct radiometry for an image**
- **PigBrtCorrModel: Implement brightness corrections for a mosaic**

Extending the Library

- **Library has been extended tremendously since initial implementation**
  - Ease of extension is one measure of quality of design
- **Coordinate systems**
  - Initial design assumed everything measured in one CS
  - Added CS tag and conversions to every vector and coordinate
    - Two months before MPL landing, took one month
- **New camera model type**
  - CAHVORE (fisheye) for MER: 1.5 weeks (math developed elsewhere at JPL)
- **Multiple Sites for rovers**
  - Added for FIDO and MER: 4 work months, very few application changes
- **Output label models**
  - Added for MER to handle different metadata style; 4 weeks including MER adaptation
- **Radiometry and brightness correction**
  - About 2 weeks each

Lessons Learned

- **Multimission design clearly successful**
  - How does one reproduce these results?
- **It helps to have more than one mission!**
  - Determining commonalities across your mission set is critical
    - Spent about 2 months initially analyzing MPF and MPL commonalities and differences
  - Understand your problem domain thoroughly
- **Develop algorithms first, then make them reusable**
  - Experience gained through MPF implementation was invaluable in getting the mission framework right
  - Unclear if "scratch" implementation of framework would have succeeded so well
  - Even if only one mission, implementing the algorithms first generates ideas for what goes in a library
    - ***Must*** have management buy-in to go fix it later!
      - All too easy to say "it works, why make it better"

Lessons Learned (cont)

- **Resist the temptation to "cheat" – even when a deadline is looming**
  - Creating abstraction layer for a new feature is harder than sneaking in mission dependencies
  - Cheating will bite you in the long run
- **Mission Designs should maintain consistency with previous missions as much as possible**
  - This is the biggie! And the toughest nut to crack
    - All missions want to do it "better" than before, and change things for no good reason
    - Example: Image labels completely redesigned for MER
      - 2 months implementation time; much more time designing debating, documenting new labels
      - Fast M01 adaptation largely due to zero label changes
    - Example: most of MSL…
      - … there's a *reason* it went over budget…

Lessons Learned (cont)

- **Don't be afraid to change your core**
  - It may require modifying dozens of applications to match, but when necessary this is cheaper than ugly workarounds
    - Overloaded functions (i.e. extra arguments) can come in handy sometimes to avoid app changes
- **Document the library extensively**
  - Javadoc-style comments directly in the code (or .h file) are critical
    - Make sure to explain the API and what mission subclasses are supposed to do
    - Do it as you go – no cheating
  - Separate documents (typical of "software development processes") are far less useful
    - Can be actively harmful if not rigorously kept up-to-date
- **Design for flexibility and extensibility**
  - The best libraries keep interfaces simple
    - Allows more applications to make use of them
  - Building blocks, not monoliths
- **Put your developers in operations**
  - They learn a *lot* about how their software is used
  - Operations benefit from in-depth knowledge of software

Conclusions

- **Must thoroughly understand your problem domain and mission set**
- **Reuse greatly reduces development costs**
  - Savings can be invested in new/improved capabilities
    - Or returned to sponsor
  - Worth the extra time to "do it right"
- **Operator training greatly reduced**
  - MIPL MER personnel can step into MSL easily because the programs are familiar
- **Application programs much easier to write**
  - Can assume core capabilities exist already